CS221
Debugging with CodeView, Visual Studio, WinDbg

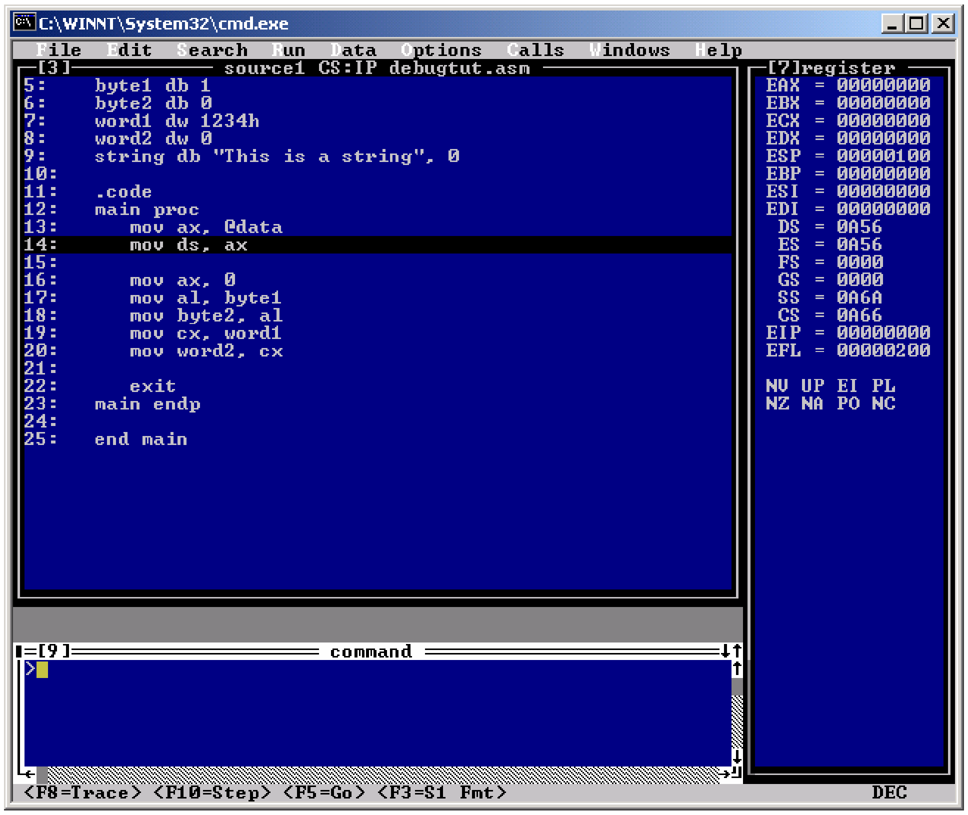
Debuggers are extremely useful tools to help you uncover errors in your program. There are different debuggers that come with MASM:

Real mode debugger: CodeView
Protected mode debugger: Visual Studio or WinDbg

Let's start by taking a closer look at using CodeView. Microsoft CodeView is the debugger that comes with MASM.

If you have followed the instructions to install MASM, you should already have CodeView set up as the debugger for 16 bit programs through the T)ools menu on TextPad. If you want to debug a program named "myfile.asm" directly from DOS then first assemble it and you can debug it using codeview by typing "**cv myfile**" (without the asm) in the same directory as the source files.

First, assemble the program and then invoke the debugger from TextPad. You should be shown with a window that appears something like that below. You may have additional windows. You can close or resize them as you wish, and open new ones from the W)indows menu.


C:\WINNT\System32\cmd.exe
File Edit Search Run Data Options Calls Windows Help
[3] source1 CS:IP debugtut.asm
[7]register
[9] command
<F8=Trace> <F10=Step> <F5=Go> <F3=S1 Fmt>
DEC

```
    byte1 db 1
    byte2 db 0
    word1 dw 1234h
    word2 dw 0
    string db "This is a string", 0

    .code
    main proc
       mov ax, @data
       mov ds, ax

       mov ax, 0
       mov al, byte1
       mov byte2, al
       mov cx, word1
       mov word2, cx

       exit
    main endp

    end main
```

```
EAX = 00000000
EBX = 00000000
ECX = 00000000
EDX = 00000000
ESP = 00000100
EBP = 00000000
ESI = 00000000
EDI = 00000000
 DS = 0A56
 ES = 0A56
 FS = 0000
 GS = 0000
 SS = 0A6A
 CS = 0A66
EIP = 00000000
EFL = 00000200

NV UP EI PL
NZ NA PO NC
```

```
>
```

I find it most useful to have the registers, source, and command windows available. You can type commands into the command window or you can also invoke most commands by selecting them from the menu bar.

I also find it helpful to put the window into full-screen mode (alt-Enter). If you wish to use graphical mode and want to use the mouse to resize windows, you may need to right-click the window, go to properties, and make sure that Insert and Quick-Edit modes are disabled.

Here is a sample program we will use to illustrate the CodeView debugger:

```
Title CodeView Tutorial Example
INCLUDE Irvine16.inc

.data
byte1 db 1
byte2 db 0
word1 dw 1234h
word2 dw 0
string db "This is a string", 0

.code
dummy proc
   mov bx, 0FFFFh
   ret
dummy endp

main proc
   mov ax, @data
   mov ds, ax

   mov ax, 0
   mov al, byte1
   mov byte2, al
   call dummy
   mov cx, word1
   mov word2, cx
   exit
main endp

end main
```

Some of the most commonly used commands are:

F5 – Execute program to the end
F8 – Step one line, go into procedure calls
F10 – Step one line, but go over procedure calls
F9 – Set or Clear a breakpoint on the cursor line

Here are some format specifiers you can use in conjunction with displaying data:

d = signed decimal integer
u = unsigned decimal integer
x = hexadecimal integer
f = floating point decimal
c = single ASCII character
s = string , terminated by NULL (0) byte

Here are some commands you can type into the Command Window:

? <expression>, <format>
    Display an expression or identifier using the above format
DB <identifier>
    Display memory from address of the identifier as bytes

DA <identifier>
    Display memory from address of the identifier as ascii
DW <identifier>
    Display memory from address of the identifier as words
DI <identifier>
    Display memory from address of the identifier as signed ints
EB <identifier>
    Enter a new byte value into identifier
EW <identifer>
    Enter a new word value into identifier
W? <identifier>, <format>
    Watch an identifier's value using the specified format
    For strings, use & in front of the identifier to get the identifier's address

Examples:

? byte1	- Displays byte1 using the default, which is decimal
? word1	- Displays word1 using the default, which is decimal
? word1, x	- Displays word1 as hex
? string	- Displays first byte of string
da string	- Displays entire string
dw string	- Displays string as groups of words
ew word1	- Enter a new word value into word1
eb byte1	- Enter a new byte value into byte1
w? word2	- Add a watch on word2
w? word2, x	- Add a watch on word2, display in hex
w? &string	- Add a watch on a string
w? &string, s	- Add a watch on a string, display as a string

For the sample program, try:

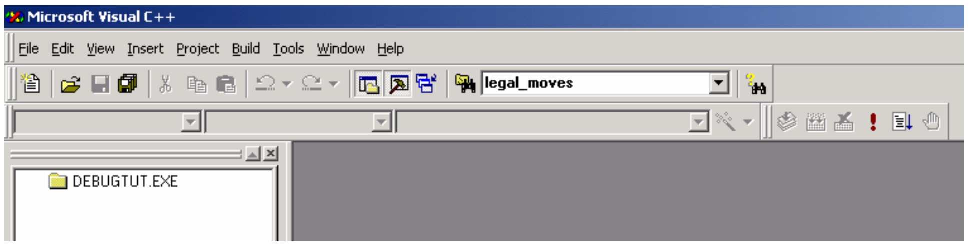
- Select O)ptions, S)ource and experiment with cha nging the view from source to mixed to machine.
- Display the registers window and resize it on the right.
- Examine variables using the ? and da commands
- Trace the program and note changes in the registers and variables using the ? commands.
- Restart the program which will reset the IP to the beginning of the program.
- From the Data menu, add byte2 and word2 as watch expressions. Re-trace the program and you should see these variables change. This is very useful for checking on programs that accidentally overwrite variables.
- Add string variables to the watch using the **w? &string** commands. Note that if you use the menu, you don't get the specify the format, so strings don't come out quite right in the watch menu.
- Add a breakpoint, restart the program, and illustrate that the program will stop execution at the breakpoint.

Debugging with Visual Studio

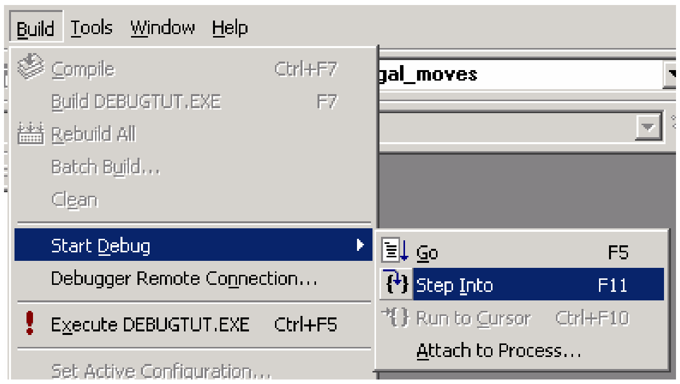
For protected mode programs, you can use the Visual Studio debugger if Visual Studio is installed on your system. Let's step through the basic features of the Visual Studio debugger using the following sample program:

```
Title Protected Mode Tutorial Example
INCLUDE Irvine32.inc

.data
byte1 db 1
byte2 db 0
word1 dw 1234h
word2 dw 0
string db "This is a string", 0

.code
dummy proc
   mov bx, 0FFFFh
   ret
dummy endp

main proc
   mov ax, 0
   mov al, byte1
   mov byte2, al
   call dummy
   mov cx, word1
   mov word2, cx
   exit
main endp

end main
```

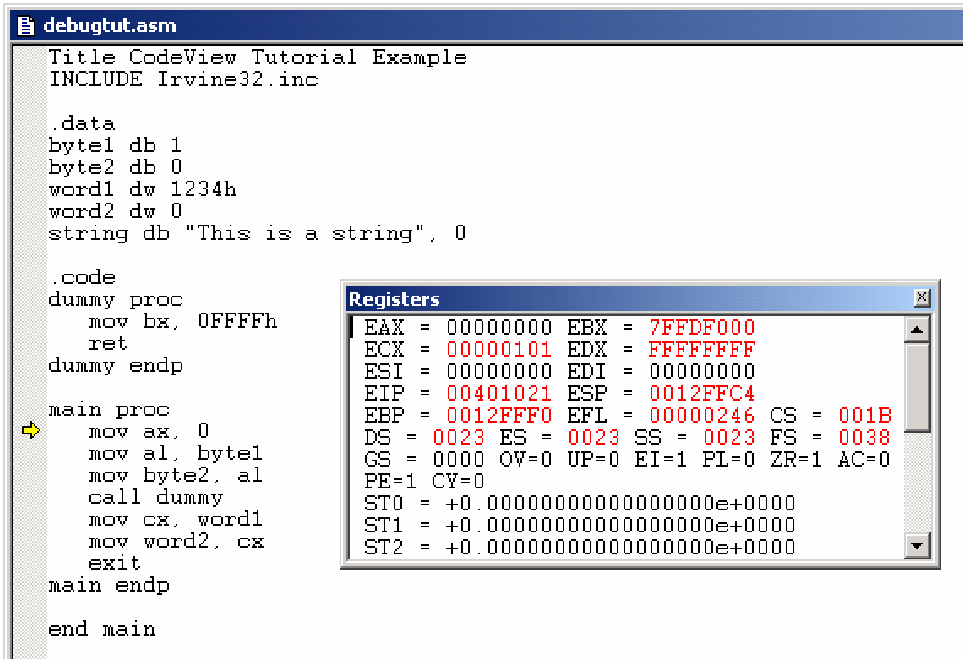
First, build the program. Then, select the debugging option from TextPad. This will launch Visual Studio and bring up an empty screen that should look somewhat like the following:



Not a very helpful screen. But if you press F11, or select B)uild, Start D)ebug, S)tep Into, then this will start the debugger and allow you to step through the program:



The debugger will show your code and stop at the first line:



debugtut.asm

```
Title CodeView Tutorial Example
INCLUDE Irvine32.inc

.data
byte1 db 1
byte2 db 0
word1 dw 1234h
word2 dw 0
string db "This is a string", 0

.code
dummy proc
   mov bx, 0FFFFh
   ret
dummy endp

main proc
   mov ax, 0
   mov al, byte1
   mov byte2, al
   call dummy
   mov cx, word1
   mov word2, cx
   exit
main endp

end main
```

Registers

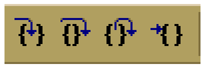
```
EAX = 00000000 EBX = 7FFDF000
ECX = 00000101 EDX = FFFFFFFF
ESI = 00000000 EDI = 00000000
EIP = 00401021 ESP = 0012FFC4
EBP = 0012FFF0 EFL = 00000246 CS = 001B
DS = 0023 ES = 0023 SS = 0023 FS = 0038
GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=1 AC=0
PE=1 CY=0
ST0 = +0.00000000000000000e+0000
ST1 = +0.00000000000000000e+0000
ST2 = +0.00000000000000000e+0000
```

The yellow arrow indicates the line of code that is about to be executed. It has not been executed yet. You will find to be most useful the contents of all the registers.

To step through the program, use:

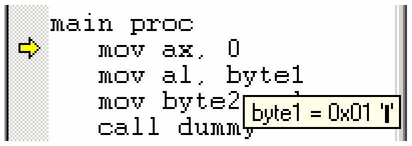
F10 - step to next instruction, but over any procedures
F11 - step to next instruction, but inside any procedures

Alternately you can use the menu or hit the icons:



The icons represent stepping into, stepping over, stepping out of, or running to the cursor location.

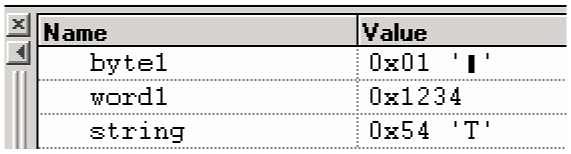
To inspect the contents of variables, at any point in time you can hover over the variable or use the watch window. If you hover the mouse over a variable, a pop-up window will display the contents of that variable. In the example below, I have hovered the mouse over the variable "byte1" and the display shows that it holds the value 1 in hex:



```
main proc
   mov ax, 0
   mov al, byte1
   mov byte2
   call dumm
```

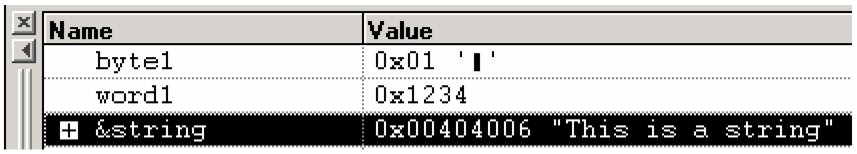
byte1 = 0x01 '▮'

To use the watch window, enter the name of the variable you would like to see. You can right-click on the Value field to show the value in either decimal or hex. In the example below, I have entered the names for byte1, word1, and string:



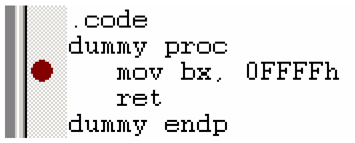
Name	Value
byte1	0x01 '▮'
word1	0x1234
string	0x54 'T'

Notice that the string variable only displays the first character. To show the whole string, enter &string. & in front of any variable will display memory starting at that address:

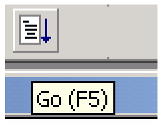


Name	Value
byte1	0x01 '▮'
word1	0x1234
⊞ &string	0x00404006 "This is a string"

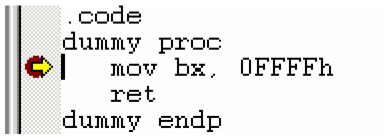
Sometimes you might not want to step through every line of code, but want the program to stop at some specific line. This is called a breakpoint. To set a breakpoint, move the cursor to the line you want execution to stop, and hit the icon with the hand on it. Alternately, you can right-click the mouse in the border of the code window. A breakpoint will be visible with a red dot. In this example, I set a breakpoint inside the dummy procedure:



```
.code
dummy proc
    mov bx, 0FFFFh
    ret
dummy endp
```

If I run the code by hitting F5 or selecting "Go" 

Then the program will halt when execution reaches this line of code:

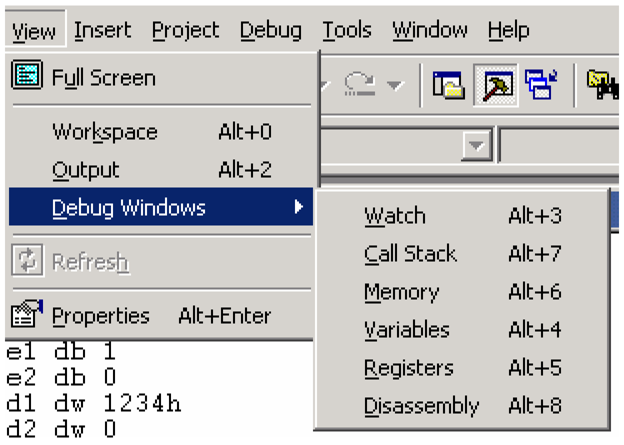


```
.code
dummy proc
    mov bx, 0FFFFh
    ret
dummy endp
```

This is indicated by the yellow arrow over the red breakpoint. At this point, you are free to inspect variables in the procedure, step through the code line by line, etc. To remove a breakpoint right-click it again or click on the Hand icon to toggle the breakpoint.

When you are through debugging, simply close Visual Studio. The program may ask if you wish to save any project information – select no unless for some reason you wish to resume this debugging session at another time. You can then continue editing your program in TextPad. If you wish to make any changes to your program in TextPad, you must make sure that any debugging sessions are closed before rebuilding your program.

There are many other options available within Visual Studio. I encourage you to explore them on your own. The other debug windows are visible from the V)iew, D)ebug Windows menu:



```
e1 db 1
e2 db 0
d1 dw 1234h
d2 dw 0
```

Debugging with WinDbg

For protected mode programs, you can use the WinDbg program, which is freely available from Microsoft. See the web page installation instructions for how to download and install WinDbg with TextPad.

Assuming that you have WinDbg installed, let's step through its basic features. WinDbg operates in a manner similar to Visual Studio's debugger.

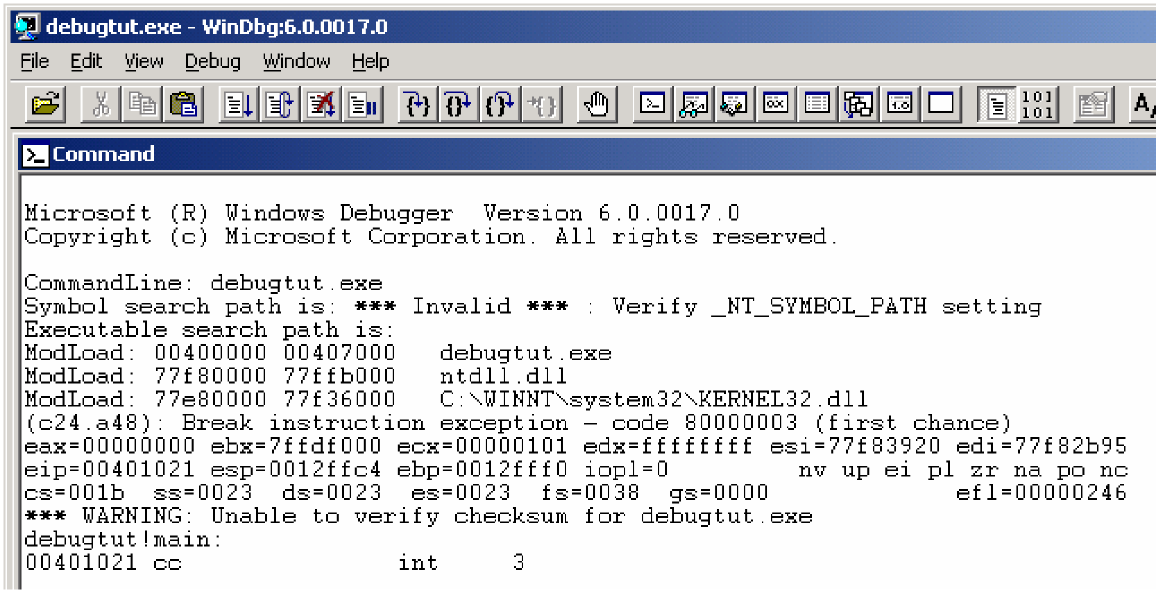
A tutorial on using WinDbg written by the author of your textbook is also available online at http://www.nuvisionmiami.com/books/asm/debug/windbg/index.htm

Let's step through WinDbg using the same sample program we had for Visual Studio:

```
Title Protected Mode Tutorial Example
INCLUDE Irvine32.inc

.data
byte1 db 1
byte2 db 0
word1 dw 1234h
word2 dw 0
string db "This is a string", 0

.code
dummy proc
   mov bx, 0FFFFh
   ret
dummy endp

main proc
   INT 3
   mov ax, 0
   mov al, byte1
   mov byte2, al
   call dummy
   mov cx, word1
   mov word2, cx
   exit
main endp

end main
```

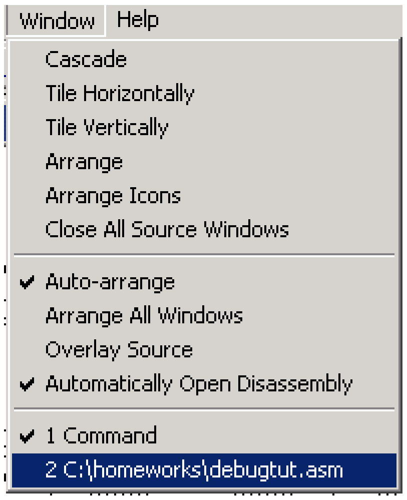
You will notice one significant difference. The line INT 3 must be added as the first line in the program. This is specific to the way WinDbg works. The instruction forces your program to halt and transfer control to the debugger.

Next, build the program. Then, select the debugging option from TextPad. This will launch WinDbg and bring up an empty screen that should look somewhat like the following:

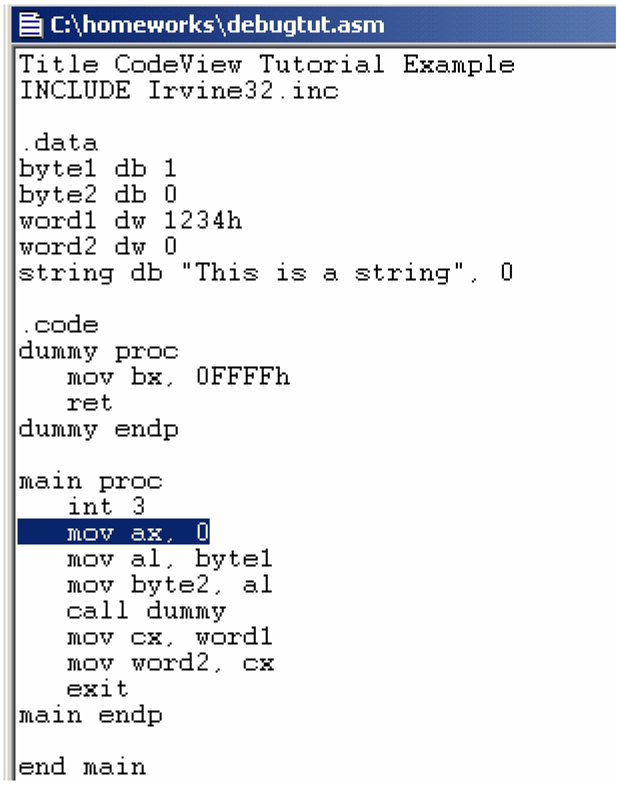

debugtut.exe - WinDbg:6.0.0017.0
File Edit View Debug Window Help
Command

```
Microsoft (R) Windows Debugger  Version 6.0.0017.0
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: debugtut.exe
Symbol search path is: *** Invalid *** : Verify _NT_SYMBOL_PATH setting
Executable search path is:
ModLoad: 00400000 00407000   debugtut.exe
ModLoad: 77f80000 77ffb000   ntdll.dll
ModLoad: 77e80000 77f36000   C:\WINNT\system32\KERNEL32.dll
(c24.a48): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=7ffdf000 ecx=00000101 edx=ffffffff esi=77f83920 edi=77f82b95
eip=00401021 esp=0012ffc4 ebp=0012fff0 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000246
*** WARNING: Unable to verify checksum for debugtut.exe
debugtut!main:
00401021 cc               int     3
```

This default view is not too useful.   Select the source code view from the menu by selecting W)indow and then the name of your source code file. In this case, my source code file is named debugtut.asm:



This will display the source code inside the debugger. The line that is bolded is where the debugger has stopped. The next instruction is the one that will be executed. In the picture below, we are about to execute the instruction “mov ax,0”:

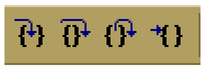


```
Title CodeView Tutorial Example
INCLUDE Irvine32.inc

.data
byte1 db 1
byte2 db 0
word1 dw 1234h
word2 dw 0
string db "This is a string", 0

.code
dummy proc
   mov bx, 0FFFFh
   ret
dummy endp

main proc
   int 3
   mov ax, 0
   mov al, byte1
   mov byte2, al
   call dummy
   mov cx, word1
   mov word2, cx
   exit
main endp

end main
```

To step through the program, use:

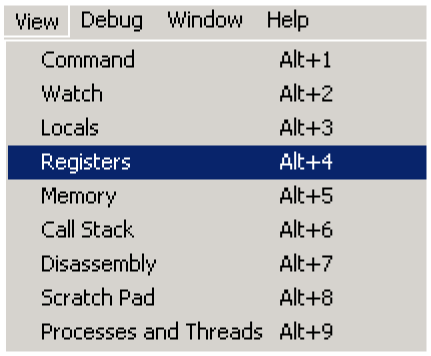
F10 - step to next instruction, but over any procedures
F11 - step to next instruction, but inside any procedures

Alternately you can use the menu or hit the icons:

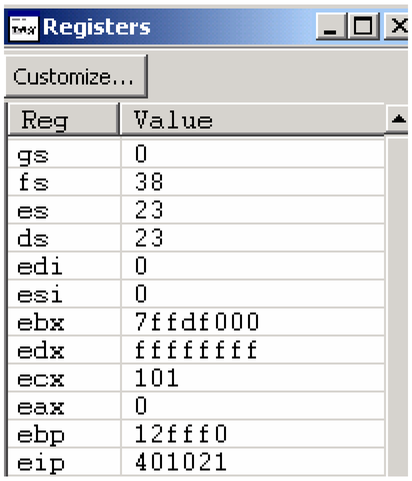


The icons represent stepping into, stepping over, stepping out of, or running to the cursor location.

To view the contents of registers, bring up the registers window. This is available from the V)iew menu:



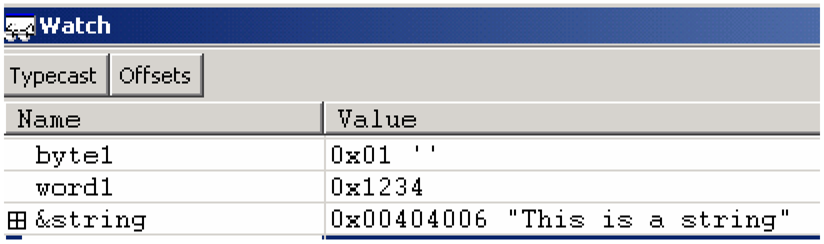
As you can see, there are several other windows available to view. We'll only talk about the Registers and the Watch window. Upon selecting the registers window, a window will appear with the contents of each register:



Reg	Value
gs	0
fs	38
es	23
ds	23
edi	0
esi	0
ebx	7ffdf000
edx	ffffffff
ecx	101
eax	0
ebp	12fff0
eip	401021

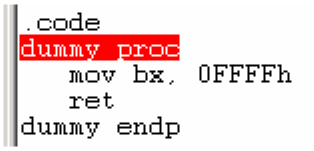
The list of registers may be much longer than what we have discussed in class. That is because this list also contains floating point and MMX/SSE registers that are available on more recent processors. As you step through the program, the registers that have changed value will be updated in red in this window.

To view the contents of variables, use the Watch window, also available from the main menu under V)iew. Enter the name of the variable you are interested in, and its contents will be displayed. To view strings, use an & in front of the string just as with Visual Studio. In the example below, we are looking at the contents of byte1, word1, and string:



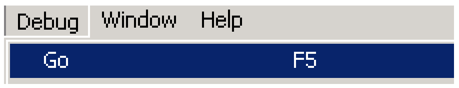
Name	Value
byte1	0x01 ''
word1	0x1234
⊞ &string	0x00404006 "This is a string"

To set a breakpoint, click the hand icon on the line that you want execution to stop. The line will turn red. In the picture below, I have set a breakpoint in the beginning of the dummy procedure:

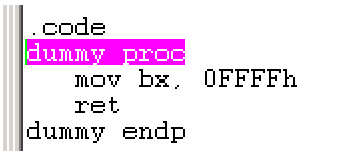


```
.code
dummy proc
    mov bx, 0FFFFh
    ret
dummy endp
```

Run the program by hitting F5 or selecting "Go" from the Debug menu:



The program will run and if it encounters a line with a breakpoint, execution will stop. In the picture below, we have halted execution in the dummy procedure as indicated by the purple highlighting:



```
.code
dummy proc
   mov bx, 0FFFFh
   ret
dummy endp
```

We can now step through the program, view any local variables, etc.

When you are finished debugging your program, simply close the WinDbg program. You can then resume editing your program in TextPad. If you wish to make any changes to your program in TextPad, you must make sure that any debugging sessions are closed before rebuilding your program.

As with Visual Studio, there are many other options available. Feel free to explore the program to see these other debugging options. Some of these are explained in more detail on the online tutorial. You may also get full instructions on WinDbg from Microsoft's MSDN library.

Conclusions

Some people are hesitant to use a debugger because of the learning curve that is involved. However, if you take the time to learn the debugger this will be a tremendous aid in tracking down errors in your program. The alternative is to scatter print statements and DUMPREGS calls throughout the program, or to try variations of the code until you can figure out what is wrong. Often these bugs can be discovered immediately with the help of a debugger.

I strongly encourage you to become familiar with a debugger for assembly and also for use in high-level languages such as Java or C++.